A Genetically Modified Hoare Logic

G. Bernot[1], J.-P. Comet[1], Z. Khalis[1], A. Richard[1], O. Roux[2]

[1] University Nice-Sophia Antipolis
I3S laboratory, UMR CNRS 7271, Les Algorithmes, bât. Euclide B, BP.121,
06903 Sophia Antipolis Cedex, France

[2] IRCCyN UMR CNRS 6597, BP 92101,
1 rue de la Noë, 44321 Nantes Cedex 3, France

Abstract: An important problem when modeling gene networks lies in the identification of parameters, even if we consider a purely discrete framework as the one of René Thomas. Here we are interested in the exhaustive search of all parameter values that are consistent with observed behaviors of the gene network. We present in this article a new approach based on Hoare Logic and on a weakest precondition calculus to generate constraints on possible parameter values. Observed behaviors play the role of "programs" for the classical Hoare logic, and computed weakest preconditions represent the sets of all compatible parameterizations expressed as constraints on parameters. Finally we give a proof of correctness of our Hoare logic for gene networks as well as a proof of completeness based on the computation of the weakest precondition.

1 Introduction

Gene regulation is a complex process where the expression level of a gene at each time depends on a large amount of interactions with related genes. Hence regulations between genes can be seen as a gene network. Different methods for studying the behavior of such gene networks in a systematic way have been proposed. Among them, ordinary differential equations played an important role which however mostly lead to numerical simulations. Moreover, the nonlinear nature of gene regulations makes analytic solutions hard to obtain. Besides, the abstraction procedure of René Thomas [Tho91], approximating sigmoid functions by step functions, makes it possible to describe the qualitative dynamics of gene networks as paths in a finite state space. Nevertheless this qualitative description of the dynamics is governed by a set of parameters which remain difficult to be deduced from classical experimental knowledge. Therefore, even when modeling with the discrete approach of René Thomas, the main difficulty lies in the identification of these parameters. In this context, we are interested in the exhaustive search of parameter values which are consistent with specifications given by the observed behavior of gene regulatory networks. Because of the exponential number of parameterizations to consider, two main kinds of approaches have emerged. On the one hand, information about cooperation or concurrence between two regulators of a same target can be taken into account in order to reduce the number of parameterizations to consider, see for example [KCRB09] and also [CTF$^+$09] in which this notion of cooperation is treated *via* a grouping of states. On the other hand, using constraints can be helpful to represent the set of consistent parameterizations see for example [FCT$^+$04, CTF$^+$09, MGCLG07].

In this paper, we present a new approach based on Hoare Logic [Hoa69] and on weakest precondition calculus [Dij75] to generate constraints on parameters. A feature of this approach lies in the fact that specifications are partially described by a set of paths, seen as "programs." Since this method avoids building the complete state graph, it results in a powerful tool to find out the constraints representing the set of consistent parameterizations with a tangible gain for

CPU time. Indeed, the weakest precondition computation which builds the constraints, goes through the "program" but is independent of the size of the gene network.

Other works were undertaken with such objectives. The application of temporal logic to biological regulatory networks was presented in [BCRG04]. Constraint programming was used for biological systems in [BPGT01] and these ideas were continued specifically for genetic regulatory networks in [Cor08, CTF$^+$09].

The paper is organized as follows. The basic concepts of Hoare logic and Dijkstra weakest precondition are quickly reminded in Section 2. The formal definitions for discrete gene regulatory networks are given in Section 3. Section 4 gives the way to describe properties of states, then presents the path language, and finally introduces the notion of Hoare triplet. The semantics of these extended Hoare triples is given in Section 5. With the previous material, in Section 6 an extended Hoare logic for gene networks is defined for Thomas' discrete models. In Section **??**, the example of the "incoherent feedforward loop of type 1" (made popular by Uri Alon in [SOMMA02, MSOI$^+$02]) highlights the whole process of our approach to find out the suitable parameter values. Section 8 contains a proof of correctness of our Hoare logic for gene networks as well as a proof of completeness based on the computation of the weakest precondition. We conclude in Section 9.

2 Reminders on standard Hoare logic

Hoare logic is a formal system for reasoning about the correctness of imperative programs. In [Hoa69], *C. A. R. Hoare* introduced the notation "$\{P\}$ *pgm* $\{Q\}$" to mean "If the assertion P (precondition) is satisfied before performing the program *pgm* and if the program terminates, then the assertion Q (postcondition) will be satisfied afterwards." This constitutes *de facto* a specification of the program under the form of a triple, called the Hoare triple.

In [Dij75], E. W. Dijkstra has defined an algorithm taking the postcondition Q and the program *pgm* as input and computing the *weakest precondition* P_0 that ensures Q if *pgm* terminates. In other words, the Hoare triple $\{P_0\}$ *pgm* $\{Q\}$ is satisfied and, for any precondition P, $\{P\}$ *pgm* $\{Q\}$ is satisfied if and only if $P \Rightarrow P_0$.

Hoare logic and weakest preconditions are now widely known and teached all over the world. The basic idea is to stamp the sequential phases of a program with assertions that are infered according to the instruction they surround. There are several equivalent versions of Hoare logic and our prefered one is the following because it offers a simple proof strategy to compute the weakest precondition *via* a proof tree. Here, p, p_1 and p_2 stand for programs, P, P_1, P_2, I and Q stand for assertions, v stands for a declared variable of the imperative program, and $Q[v \leftarrow expr]$ means that *expr* is substituted to each free occurrence of v in Q.

Assignment rule: $$\frac{}{\{Q[v{\leftarrow}expr]\}\ v{:=}expr\ \{Q\}}$$

Sequential composition rule: $$\frac{\{P_2\}\ p_2\ \{Q\} \qquad \{P_1\}\ p_1\ \{P_2\}}{\{P_1\}\ p_1;p_2\ \{Q\}}$$

Alternative rule: $$\frac{\{P_1\}\ p_1\ \{Q\} \qquad \{P_2\}\ p_2\ \{Q\}}{\{(e{\wedge}P_1){\vee}(\neg e{\wedge}P_2)\}\ \texttt{if}\ e\ \texttt{then}\ p_1\ \texttt{else}\ p_2\ \{Q\}}$$

Iteration rule: $$\frac{\{e{\wedge}I\}\ p\ \{I\}}{\{I\}\ \texttt{while}\ e\ \texttt{with}\ I\ \texttt{do}\ p\ \{\neg e{\wedge}I\}}$$

Empty program rule: $$\frac{P{\Rightarrow}Q}{\{P\}\ \varepsilon\ \{Q\}}$$ (where ε stands for the empty program)

There are standard additional rules: first order logic (to establish "$P \Rightarrow Q$" introduced by the Empty program rule), and in practice some reasonnings about the data structures of the

program (e.g. integers) in order to simplify the expressions as much as possible "on the fly" in the proof trees.

The *Iteration rule* requires some comments. The assertion I is called the *loop invariant* and it is well known that finding the weakest loop invariant is undecidable [Hat74, BG01]. It has been included within the programming language for this reason; we ask the programmer to give a loop invariant explicitely after the `with` keyword, although it may appear redundant as it is also the precondition of the Hoare triple. By doing so, within a program, each `while` instruction carries its own (sub)specification and it can consequently be proved apart from the rest of the program.

When using these Hoare logic rules, the following proof strategy builds a proof tree that performs the proof by computing the weakest precondition [Dij75]:

1. For each `while` statement within a Hoare triple

   $$\{P\}\ p_1\ ;\ \texttt{while}\ e\ \texttt{with}\ I\ \texttt{do}\ p_2\ ;\ p_3\ \{Q\}$$

   perform 3 independent sub-proofs:

   - $\{\neg e \wedge I\}\ p_3\ \{Q\}$
   - $\{I\}$ `while` e `with` I `do` $p_2\ \{\neg e \wedge I\}$ (i.e. $\{e \wedge I\}\ p_2\ \{I\}$ according to the Iteration rule)
   - $\{P\}\ p_1\ \{I\}$

   This first step of strategy leads to proofs on subprograms that do not contain any `while` instruction.

2. Apply the *Sequential composition rule* only when the program p_2 of this rule is reduced to an instruction, which leads to perform the proof starting from postcondition Q at the end and treat the instructions backward.

3. Never apply the *Empty program rule*, except when the leftmost instruction has been treated, that is when all instructions have been treated.

Since the *Assignment rule*, which is central, makes it possible to precisely define *one* precondition from its postcondition, and since the other rules relate (but do not evaluate) the conditions, the proof tree is done from the end to the beginning of the program and computes a unique assertion. By doing so, the precondition obtained just before applying the last Empty program rule is actually the weakest precondition (assuming that the programmer has given the weakest loop invariants). In the remainder of the article, we call this strategy *the backward strategy*. In Section **??**, we always follow the backward strategy.

The most striking feature of Hoare logic and weakest precondition is that proofs according to the backward strategy consist in very simple sequences of syntactic formula substitutions and end with first order logic proofs. Nevertheless, it is worth noticing that it is only a question of *partial* correctness since Hoare Logic does not give any proof of the termination of the analyzed program (`while` instructions may induce infinite loops).

3 Discrete gene regulatory networks with multiplexes

This section presents our modeling framework, based on the general discrete method of René Thomas [Td90, TK01] and introduced in [KCRB09, Kha10].

The starting point consists in a labeled directed graph in which vertices are either *variables* or *multiplexes*. Variables abstract genes or their products, and multiplexes contain propositional formulas that encode situations in which a group of variables (inputs of multiplexes) influence the evolution of some variables (outputs of multiplexes). Hence multiplexes represent biological phenomena, such as the formation of complexes to activate some genes. In the next definition, this labeled directed graph is formally defined, and it is associated with a family $\mathcal{K}$ of integers. As we will see later, these integers correspond to parameters that drive the dynamics of the network.

Definition 3.1 *A* gene regulatory network with multiplexes *(*GRN *for short) is a tuple* $N = (V, M, E_V, E_M, \mathcal{K})$ *satisfying the following conditions:*

- *V and M are disjoint sets, whose elements are called* variables *and* multiplexes *respectively.*
- *$\tilde{N} = (V \cup M, E_V \cup E_M)$ is a labeled directed graph such that:*
  - *edges of E_V start from a variable and end to a multiplex, and edges of E_M start from a multiplex and end to either a variable or a multiplex.*
  - *every directed cycle of $\tilde{N}$ contains at least one variable.*
  - *every variable v of V is labeled by a positive integer b_v called the* bound *of v.*
  - *every multiplex m of M is labeled by a formula φ_m belonging to the language $\mathcal{L}_m$ inductively defined by:*
    - *If $v \to m$ belongs to E_V and $s \in \mathbb{N}$ belongs to the interval $[1, b_v]$, then $v \geq s$ is an atom of $\mathcal{L}_m$.*
    - *If $m' \to m$ belongs to E_M then m' is an atom of $\mathcal{L}_m$.*
    - *If φ and ψ belong to $\mathcal{L}_m$ then $\neg\varphi$, $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$ also belong to $\mathcal{L}_m$.*
- *$\mathcal{K} = \{K_{v,\omega}\}$ is a family of integers indexed by $v \in V$ and $\omega \subset N^{-1}(v)$, where $N^{-1}(v)$ is the set of predecessors of v in $\tilde{N}$ (that is, the set of multiplexes m such that $m \to v$ is an edge of E_M). Each $K_{v,\omega}$ must satisfy $0 \leq K_{v,\omega} \leq b_v$.*

Notation 3.2 *The* flaten version *of a formula φ_m, denoted $\overline{\varphi_m}$, is obtained by applying the following algorithm: while the formula contains a multiplex atom m', substitute m' by its associated formula $\varphi_{m'}$. The formula $\overline{\varphi_m}$ exists since $\tilde{N}$ has no directed cycle with only multiplexes. As a result, all the atoms of $\overline{\varphi_m}$ are of the form $v \geq s$.*

A *state* is an assignment of integer values to the variables of V. Such an assignment allows a natural evaluation of any formula φ_m: by replacing variables by their values, $\overline{\varphi_m}$ becomes a propositional formula whose atoms are integer inequalities.

Definition 3.3 (states, satisfaction relation and resources). *Let N be a* GRN *and V be its set of variables. A* state *of N is a function $\eta : V \to \mathbb{N}$ such that $\eta(v) \leq b_v$ for all $v \in V$. Let $\mathcal{L}$ be the set of propositional formula whose atoms are of the form $v \geq s$ with $v \in V$ and s be a positive integer (so that $\overline{\varphi_m}$ is a formula of $\mathcal{L}$ for every multiplex m of N). The satisfaction relation $\models_N$ between a state η of N and a formula φ of $\mathcal{L}$ is inductively defined by:*

- *if φ is reduced to an atom of the form $v \geq s$, then $\eta \models_N \varphi$ if and only if $\eta(v) \geq s$.*
- *if $\varphi \equiv \psi_1 \wedge \psi_2$ then $\eta \models_N \varphi$ if and only if $\eta \models_N \psi_1$ and $\eta \models_N \psi_2$; and we proceed similarly for the other connectives.*

Given a variable $v \in V$, a multiplex $m \in N^{-1}(v)$ is a resource *of v at state η if $\eta \models_N \overline{\varphi_m}$. The* set of resources *of v at state η is defined by $\rho(\eta, v) = \{m \in N^{-1}(v) \mid \eta \models_N \overline{\varphi_m}\}$.*

From a dynamical point of view, at a given state η, each variable v is supposed to evolve in the direction of a specific level (between 0 and b_v) that only depends on the set $\rho(\eta, v)$. This *focal level* is given by the *logical parameter* $K_{v,\rho(\eta,v)}$. Hence, at state η, v can increase if $\eta(v) < K_{v,\rho(\eta,v)}$, it can decrease if $\eta(v) > K_{v,\rho(\eta,v)}$, and it is stable if $\eta(v) = K_{v,\rho(\eta,v)}$.

Suppose for instance that v has two input multiplexes m_{ab} and m_{cd} with formula $(a \geq 1 \wedge b \geq 1)$ and $(c \geq 1 \wedge d \geq 1)$ respectively. Then m_{ab} and m_{cd} may be seen as complexes (dimers) regulating the level of v. Suppose, in addition, that $K_{v,\emptyset} = 0$, $K_{v,\{m_{ab}\}} = K_{v,\{m_{cd}\}} = 1$ and $K_{v,\{m_{ab},m_{cd}\}} = 2$. Then, complexes m_{ab} and m_{cd} specify activator complexes, with an individual effect which is less than a cumulated effect (the focal level of v in the presence of a single complex is less than the focal level of v in the presence of both complexes). This example illustrates the fact that multiplexes encode combinations of variables that regulate a given variable, and that the parameters, by giving a weight to each possible combinations of multiplexes, indicate how multiplexes regulate a given variable.

As in Thomas' method [Td90, TK01], it is assumed that variables evolve asynchronously and by unit steps toward their respective target levels. The dynamics of a gene regulatory network is then described by the following asynchronous state graph.

Definition 3.4 (State Graph). *Let $N = (V, M, E_V, E_M, \mathcal{K})$ be a* GRN. *The* state graph *of N is the directed graph $\mathcal{S}$ defined as follows: the set of vertices is the set of states of N, and there exists an edge (or transition) $\eta \to \eta'$ if one of the following conditions is satisfied:*

- *there is no $v \in V$ such that $\eta(v) \neq K_{v,\rho(\eta,v)}$ and $\eta' = \eta$.*
- *there exists $v \in V$ such that $\eta(v) \neq K_{v,\rho(\eta,v)}$ and*

$$\eta'(v) = \begin{cases} \eta(v) + 1 & \text{if } \eta(v) < K_{v,\rho(\eta,v)} \\ \eta(v) - 1 & \text{if } \eta(v) > K_{v,\rho(\eta,v)} \end{cases} \qquad \text{and} \qquad \forall u \neq v,\ \eta'(u) = \eta(u).$$

Hence a state η is a stable state if and only if it has itself as successor, that is, if and only if every variable is stable at state η (*i.e.* $\eta(v) = K_{v,\rho(\eta,v)}$ for every variable v). If η is not a stable state, then it has at least one outgoing transition. More precisely, for each variable v such that $\eta(v) \neq K_{v,\rho(\eta,v)}$, there is a transition allowing v to evolve (± 1) toward its focal level $K_{v,\rho(\eta,v)}$. Every outgoing transition of η is supposed to be possible, so that there is an indeterminism as soon as η has several outgoing transitions. An example is given in Figure 2 (see also Section 7 for another example).

4 Pre- and post-conditions on path sets

In order to formalize known information about a gene network, we introduce in this section a language to express properties of states (assertion language) and a language to express properties of state transitions (path language). Combining properties of state transitions and properties of states, at the beginning and at the end of a sequences of state transitions, leads to the notion of Hoare triplet on path programs.

4.1 An assertion language for discrete models of gene networks

To describe properties of states in a meaningful way, we need terms that allow us to check, compare and manipulate variable values while taking parameter values into account. The following definitions define a language suitable for such needs. It extends [Kha10].

Definition 4.1 (Terms of the assertion language) *Let $N = (V, M, E_V, E_M, \mathcal{K})$ be a* GRN. *The* well formed terms of the assertion language *of N are inductively defined by:*

- *Each integer $n \in \mathbb{N}$ constitutes a well formed constant term*
- *For each variable $v \in V$, the name of the variable v, considered as a symbol, constitutes a well formed constant term.*
- *Similarly, for each $v \in V$ and for each subset ω of $N^{-1}(v)$, the symbol $K_{v,\omega}$ constitutes a well formed constant term.*
- *If t and t' are well formed terms then $(t + t')$ and $(t - t')$ are also well formed terms.*

Definition 4.2 (Assertion language and its semantics) *Let $N = (V, M, E_V, E_M, \mathcal{K})$ be a* GRN. *The* assertion language *of N is inductively defined as follows:*

- *If t and t' are well formed terms then $(t = t')$, $(t < t')$, $(t > t')$, $(t \leq t')$ and $(t \geq t')$ are atoms of the assertion language.*
- *If φ and ψ belong to the assertion language then $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\varphi \Rightarrow \psi)$ also belong to the assertion language.*

A state η of the network N satisfies *an assertion φ if and only if its interpretation is valid in $\mathbb{Z}$, after substituting each variable v by $\eta(v)$ and each symbol $K_{v,\omega}$ by its value according to the family $\mathcal{K}$. We note $\eta \models_N \varphi$.*

4.2 A path language for discrete models of gene networks

The assertion language introduced above is a subset of first order logic well suited to describe properties on sets of states. It does not express dynamical aspects, since the dynamics of the system is encoded in the transitions of the state graph. A description of dynamical properties equates to a precise formulation of properties of paths. The language proposed here is suitable for encoding such properties.

Definition 4.3 (Path language and path program) *Let $N = (V, M, E_V, E_M, \mathcal{K})$ be a* GRN. *The* path language *of N is the language inductively defined by:*

- *For each $v \in V$ and $n \in \mathbb{N}$ the expressions "$v+$", "$v-$" and "$v := n$" belong to the path language (respectively increase, decrease or assignment of variable value).*
- *If e is a formula belonging to the assertion language of N, then "$assert(e)$" also belongs to the path language.*
- *If p_1 and p_2 belong to the path language then $(p_1; p_2)$ also belongs to the path language (sequential composition). Moreover the sequential composition is associative, so that we can write $(p_1; p_2; \cdots; p_n)$ without intermediate parentheses.*
- *If p_1 and p_2 belong to the path language and if e is a formula belonging to the assertion language of N, then $(if\ e\ then\ p_1\ else\ p_2)$ also belongs to the path language.*
- *If p belongs to the path language and if e and I are formulas belonging to the assertion language of N, then $(while\ e\ with\ I\ do\ p)$ also belongs to the path language. The assertion I is called the invariant of the while loop.*
- *If p_1 and p_2 belong to the path language then $\forall(p_1, p_2)$ and $\exists(p_1, p_2)$ also belong to the path language (quantifiers). Moreover the quantifiers are associative and commutative, so that we can write $\forall(p_1, p_2, \cdots, p_n)$ and $\exists(p_1, p_2, \cdots, p_n)$ as useful abbreviations.*

For technical purposes, we also consider the empty program *"ε" (outside the inductive definition). A well formed expression in the path language is called a* path program.

Intuitivelly, "$v+$" (resp. "$v-$", "$v := n$") means that the level of variable v is increasing by one unit (resp. decreasing by one unit, set to a particular value n). "$assert(e)$" allows one to express a property of the current state without change of state. The sequential composition allows one to concatenate two path programs whereas the statement "if" allows one to choose between two programs according to the evaluation of the formula e. Finally it becomes possible to express properties of several paths thanks to the quantifiers $\forall$ and $\exists$. Lastly notice that ε appears in a path program if and only if the path is reduced to the empty program. These intuitions will be formalized in Section 5.

4.3 Syntax of pre- and post-conditions on path programs

The next step is to combine properties of state transitions (path program) and properties of states (assertions), at the begining and at the end of the considered path program. This is done *via* the notion of Hoare triplet on path programs.

Notation 4.4 *A* GRN *N being given, a* Hoare triple *on path programs is an expression of the form "$\{P\}\ p\ \{Q\}$" where P and Q are well formed assertions, called pre- and post-condition respectively, and p is a path program.*

Intuitively, the precondition P describes a set of states, e.g. all states for which variable v has value zero ($P \equiv v = 0$), the path program p describes dynamical processes, e.g. increase of variable v ($p \equiv v+$), and the postcondition again describes a set of states, e.g. all states for which variable v has value one ($Q \equiv v = 1$). This small example encodes the process of variable v changing its value from zero to one. Whether or not the expression is satisfied for a given gene network N depends on its state transition graph, thus it depends on the corresponding parameter values in $\mathcal{K}$.

5 Semantics of Hoare triples on path programs

We firstly define the semantics of path programs *via* a binary relation. The general ideas that motivate the definition below are the following:

- Starting from an initial state η, sequences of instructions without existential or universal quantifier either transform η into another state η' or is not feasible so that η' is undefined. For example, the simple instruction $v+$ transforms η into η' (where $\forall u \neq v, \eta'(u) = \eta(u)$ and $\eta'(v) = \eta(v) + 1$) if $\eta \to \eta'$ exists. If, on the contrary, this transition does not exist, the instruction is not feasible.
- Existential quantifiers induce a sort of "non determinism" about η': according to the chosen path under each existential quantifier one may get differents resulting states. Consequently, one cannot define the semantics as a partial function that associates a unique η' to η; a binary relation "$\eta \rightsquigarrow ...$" is a more suited mathematical object.
- Universal quantifiers induce a sort of "solidarity" between all the states η' that can be obtained according to the chosen path under each universal quantifier: all of them will have to satisfy the postcondition later on. For this reason, we define a binary relation that associates a *set of states* E to the initial state η: "$\eta \rightsquigarrow E$". Such a set E can be understood as grouping together the states it contains, under the scope of some universal quantifier.

- When the path program p contains both existential and universal quantifiers, we may consequently get several sets $E_1, \cdots, E_n$ such that "$\eta \overset{p}{\rightsquigarrow} E_i$", each of the E_i being a possibility through the existential quantifiers of p and all the states belonging to a given E_i being together through the universal quantifiers of p. On the contrary, if p is not feasible, then there is no set E at all such that "$\eta \overset{p}{\rightsquigarrow} E$"

Notation 5.1 *For a state η, a variable v and $k \in [0, b_v]$, we define $\eta[v \leftarrow k]$ as the state η' such that $\eta'(v) = k$ and for all $u \neq v$, $\eta'(u) = \eta(u)$.*

Definition 5.2 [Path program relation $\overset{p}{\rightsquigarrow}$].

Let $N = (V, M, E_V, E_M, \mathcal{K})$ be a GRN*, let $\mathcal{S}$ be the state graph of N whose set of vertices is denoted S and let p be a path program of N. The binary relation $\overset{p}{\rightsquigarrow}$ is the smallest subset of $S \times \mathcal{P}(S)$ such that, for any state η:*

1. *If p is reduced to the instruction $v+$ (resp. $v-$), then let us consider $\eta' = \eta[v \leftarrow (\eta(v) + 1)]$ (resp. $\eta' = \eta[v \leftarrow (\eta(v) - 1)]$): if $\eta \to \eta'$ is a transition of $\mathcal{S}$ then $\eta \overset{p}{\rightsquigarrow} \{\eta'\}$*
2. *If p is reduced to the instruction $v := k$, then $\eta \overset{p}{\rightsquigarrow} \{\eta[v \leftarrow k]\}$*
3. *If p is reduced to the instruction $assert(e)$, if $\eta \models_N e$, then $\eta \overset{p}{\rightsquigarrow} \{\eta\}$*
4. *If p is of the form $\forall(p_1, p_2)$: if $\eta \overset{p_1}{\rightsquigarrow} E_1$ and $\eta \overset{p_2}{\rightsquigarrow} E_2$ then $\eta \overset{p}{\rightsquigarrow} (E_1 \cup E_2)$*
5. *If p is of the form $\exists(p_1, p_2)$: if $\eta \overset{p_1}{\rightsquigarrow} E_1$ then $\eta \overset{p}{\rightsquigarrow} E_1$,* and *if $\eta \overset{p_2}{\rightsquigarrow} E_2$ then $\eta \overset{p}{\rightsquigarrow} E_2$*
6. *If p is of the form $(p_1; p_2)$: if $\eta \overset{p_1}{\rightsquigarrow} F$ and if $\{E_e\}_{e \in F}$ is a F-indexed family of state sets such that $e \overset{p_2}{\rightsquigarrow} E_e$, then $\eta \overset{p}{\rightsquigarrow} (\bigcup_{e \in F} E_e)$*
7. *If p is of the form $(if\ e\ then\ p_1\ else\ p_2)$:*
   - *if $\eta \models_N e$ and $\eta \overset{p_1}{\rightsquigarrow} E$ then $\eta \overset{p}{\rightsquigarrow} E$*
   - *if $\eta \not\models_N e$ and $\eta \overset{p_2}{\rightsquigarrow} E$ then $\eta \overset{p}{\rightsquigarrow} E$*
8. *If p is of the form $(while\ e\ with\ I\ do\ p_0)$:*
   - *if $\eta \models_N e$ and $\eta \overset{p_0;p}{\rightsquigarrow} E$ then $\eta \overset{p}{\rightsquigarrow} E$*
   - *if $\eta \not\models_N e$ then $\eta \overset{p}{\rightsquigarrow} \{\eta\}$*
9. *If p is the empty program ε, then $\eta \overset{p}{\rightsquigarrow} \{\eta\}$*

This definition calls for several comments.

The relation $\overset{p}{\rightsquigarrow}$ exists because (i) the set of all relations that satisfy the properties 1–8 of the definition is not empty (the relation which links all states to all sets of states satisfies the properties) and (ii) the intersection of all the relations that satisfy the properties 1–8, also satisfies the properties.

A simple instruction such as $v+$ can be not feasible from a state η (if $\eta \to \eta'$ is not a transition of $\mathcal{S}$). In this case, there is no set E such that $\eta \overset{v+}{\rightsquigarrow} E$. The same situation happens when the program is an assertion that is evaluated to false at the current state η.

Universal quantifiers "propagate" non feasible paths: if one of the p_i is not feasible then $\forall(p_1, \cdots, p_n)$ is not feasible. *It is not the case for existential quantifiers*: if $\eta \overset{p_i}{\rightsquigarrow} E_i$ for one of the p_i then $\eta \overset{\exists(p_1 \cdots p_n)}{\rightsquigarrow} E_i$ even if one of the p_j is not feasible.

When a *while* loop does not terminate, there does not exist a set E such that $\eta \overset{while\ldots}{\rightsquigarrow} E$. This is due to the minimality of the binary relation $\overset{p}{\rightsquigarrow}$. On the contrary, when the *while* loop

terminates, it is equivalent to a program containing a finite number of the sub-program p_0 in sequence, starting from η.

The semantics of sequential composition may seem unclear for whom is not familiar with commutations of quantifiers. We better take an example to explain the construction of $\overset{p_1;p_2}{\rightsquigarrow}$ (see Figure 1):

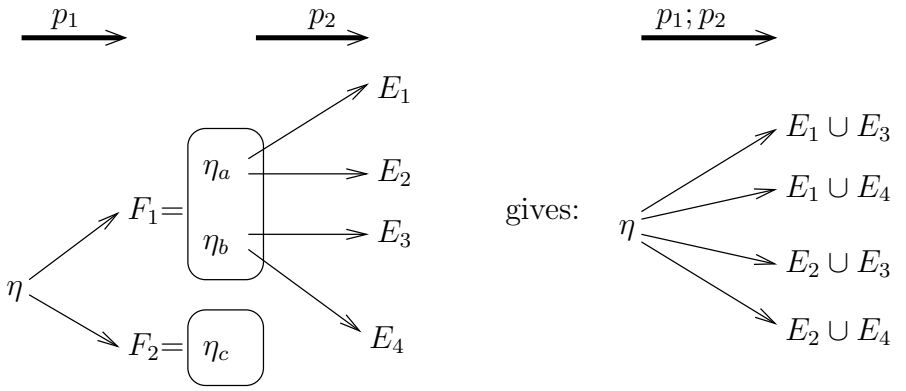

Figure 1: An example for the semantics of sequential composition

- Let us assume that starting from state η, two sets of states are reachable *via* p_1: $\eta \overset{p_1}{\rightsquigarrow} F_1 = \{\eta_a, \eta_b\}$ and $\eta \overset{p_1}{\rightsquigarrow} F_2 = \{\eta_c\}$. It intuitively means that p_1 permits a choice between F_1 and F_2 through some existential quantifier on paths and that the path leading to F_1 contains a universal quantifier grouping together η_a and η_b.
- Let us also assume that:
  - starting from state η_a, two sets of states are reachable *via* p_2: $\eta_a \overset{p_2}{\rightsquigarrow} E_1$ and $\eta_a \overset{p_2}{\rightsquigarrow} E_2$,
  - starting from state η_b, two sets of states are reachable *via* p_2: $\eta_b \overset{p_2}{\rightsquigarrow} E_3$ and $\eta_b \overset{p_2}{\rightsquigarrow} E_4$,
  - there is not any set E such that $\eta_c \overset{p_2}{\rightsquigarrow} E$

When focusing on the paths of $(p_1; p_2)$ that encounter F_1 after p_1, the paths such that p_1 leads to η_a must be grouped together with the ones that leads to η_b. Nevertheless, for each of them, p_2 permits a choice: between E_1 or E_2 for η_a and between E_3 or E_4 for η_b. Consequently, when grouping together the possible futures of η_a and η_b, one needs to consider the four possible combinations: $\eta \overset{p_1;p_2}{\rightsquigarrow} (E_1 \cup E_3)$, $\eta \overset{p_1;p_2}{\rightsquigarrow} (E_1 \cup E_4)$ $\eta \overset{p_1;p_2}{\rightsquigarrow} (E_2 \cup E_3)$ and $\eta \overset{p_1;p_2}{\rightsquigarrow} (E_2 \cup E_4)$.
Lastly, when focusing on the paths of $(p_1; p_2)$ that encounter F_2 after p_1, since η_c has no future *via* p_2, there is no family indexed by F_2 as mentioned in the definition and consequently it adds no relation into $\overset{p_1;p_2}{\rightsquigarrow}$.

Lastly, let us remark that, if $\eta \overset{p}{\rightsquigarrow} E$ then E cannot be empty; it always contains at least one state. The proof is easy by structural induction of the program p (using the fact that a *while* loop which terminates is equivalent to a program containing a finite number of the sub-program p_0).

Definition 5.3 (Semantics of a Hoare triple). *Let $N = (V, M, E_V, E_M, \mathcal{K})$ be a* GRN *and let $\mathcal{S}$ be the state graph of N whose set of vertices is denoted S. A Hoare triple $\{P\}\ p\ \{Q\}$ is satisfied if and only if:*

for all $\eta \in S$ satisfying P, there exists E such that $\eta \overset{p}{\rightsquigarrow} E$ and for all $\eta' \in E$, η' satisfies Q.

The previous definition implies the consistency of all the paths described by the path program p with the state graph: if path program p is not feasible from one of the states satisfying pre-condition P, then the Hoare triplet cannot be satisfied. For instance if some $v+$ is required by

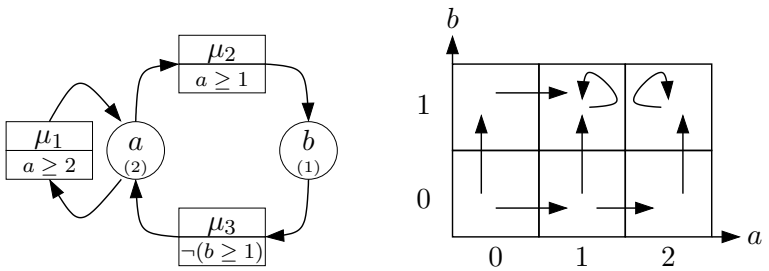

Figure 2: **(Left)** Graphical representation of the GRN $N = (V, M, E_V, E_M, \mathcal{K})$ with $V = \{a, b\}$, the bounds of a and b are respectivelly 2 and 1, $M = \{\mu_1, \mu_2, \mu_3\}$, ϕ_{μ_1} is $(a \geq 2)$, ϕ_{μ_2} is $(a \geq 1)$, ϕ_{μ_3} is $\neg(b \geq 1)$. Finally the family of integers is $\{K_{a,\emptyset} = 1,\ K_{a,\{\mu_1\}} = 1,\ K_{a,\{\mu_3\}} = 2,\ K_{a,\{\mu_1,\mu_3\}} = 2,\ K_{b,\emptyset} = 1,\ K_{b,\{\mu_2\}} = 1\}$. **(Right)** Representation of its state graph.

the path program p but the increasing of v is not possible according to the state graph, then the Hoare triple is not satisfied.

More generally the path language plays the role of the programming language in the classical Hoare Logic. A Hoare triplet is satisfied iff from the states satisfying the precondition, the program p is feasible and leads to a set of states where the postcondition is satisfied. The path program p can then be viewed as the sequence of actions one can use in order to modify the state (memory) of the system.

Nevertheless, similarly to classical Hoare logic which reflects a partial correctness of imperative programming language, the previous definition does not imply termination of *while* loops. Our path language can also define some infinite paths. Notice that if the non terminating *while* loop is at the end of the program, then it has a biological meaning: it represents periodic behaviours (such as the circadian cycle for instance).

Examples. Let us consider the GRN of Figure 2 and its state graph.

1. The Hoare triplet $\{(a = 0) \wedge (b = 0)\}\ a+; a+; b+\ \{(a = 2) \wedge (b = 1)\}$ is satisfied, because :
   - There is a unique state satisfying the precondition $(a = 0) \wedge (b = 0)$ and
   - from this state, the path program $a+; a+; b+$ is possible and leads to the state $(2, 1)$ and
   - the state $(2, 1)$ satisfies the postcondition $(a = 2) \wedge (b = 1)$.
2. On the opposite, the Hoare triplet $\{(a = 2) \wedge (b = 0)\}\ b+; a-; a-\ \{(a = 0) \wedge (b = 1)\}$ is not satisfied because from the state satisfying the precondition, the first "instruction" $b+$ is possible and leads to the state $(2, 1)$ from which the next instruction $b-$ is not consistant with the state graph.
3. The following Hoare triplet contains two existantial quantifiers and a universal one : $\{(a = 0) \wedge (b = 0)\}\ \forall(a+, b+); \exists(a+, b+); \exists(\varepsilon, b+)\ \{(b = 1)\}$
   - We have clearly $(0, 0) \overset{\forall(a+,b+)}{\rightsquigarrow} \{(1, 0), (0, 1)\}$
   - Since $(1, 0) \overset{\exists(a+,b+)}{\rightsquigarrow} \{(2, 0)\}$ and $(1, 0) \overset{\exists(a+,b+)}{\rightsquigarrow} \{(1, 1)\}$ and $(0, 1) \overset{\exists(a+,b+)}{\rightsquigarrow} \{(1, 1)\}$, we have both $(0, 0) \overset{\forall(a+,b+);\exists(a+,b+)}{\rightsquigarrow} \{(1, 1), (2, 0)\}$ and $(0, 0) \overset{\forall(a+,b+);\exists(a+,b+)}{\rightsquigarrow} \{(1, 1)\}$.
   - We have trivially $(1, 1) \overset{\exists(\varepsilon,b+)}{\rightsquigarrow} \{(1, 1)\}$
   - Moreover we have both $(2, 0) \overset{\exists(\varepsilon,b+)}{\rightsquigarrow} \{(2, 0)\}$ and $(2, 0) \overset{\exists(\varepsilon,b+)}{\rightsquigarrow} \{(2, 1)\}$

- We deduce that the considered program p can lead to 3 differents set of states : $(0,0) \overset{p}{\rightsquigarrow} \{(1,1),(2,0)\}$, $(0,0) \overset{p}{\rightsquigarrow} \{(1,1)\}$ and $(0,0) \overset{p}{\rightsquigarrow} \{(1,1),(2,1)\}$.

Because the postcondition is satisfied in both states $(1,1)$ and $(2,1)$, see the last set of states which is in relation with $(0,0)$, one can deduce that the Hoare Triplet is satisfied.

6 A Hoare logic for discrete models of gene networks

In this section, we define a "genetically modified" Hoare logic by giving the rules for each instruction of our path language (definition 4.3). First, let us introduce a few notations for intensively used formulas.

Notation 6.1 *Let $N = (V, M, E_V, E_M, \mathcal{K})$ be a* GRN *and let v be a variable of V.*

1. *For each subset ω of $N^{-1}(v)$ (set of predecessors of v in the network), we denote by Φ_v^ω the following formula:*

$$\Phi_v^\omega \quad \equiv \quad \Big(\bigwedge_{m \in \omega} \overline{\varphi_m}\Big) \wedge \Big(\bigwedge_{m \in N^{-1}(v)\setminus\omega} \neg\overline{\varphi_m}\Big)$$

where $N^{-1}(v) \setminus \omega$ stands for the complementary subset of ω in $N^{-1}(v)$.
From Definition 3.3, for all states η and for all variables $v \in V$, $\eta \models_N \Phi_v^\omega$ if and only if $\omega = \rho(\eta, v)$, that is, ω is the set of resources of v at state η. Consequently, there exists a unique *ω such that $\eta \models_N \Phi_v^\omega$.*

2. *We denote by Φ_v^+ the following formula:*

$$\Phi_v^+ \quad \equiv \quad \bigwedge_{\omega \subset G^{-1}(v)} (\Phi_v^\omega \Longrightarrow K_{v,\omega} > v)$$

From Definition 3.4, $\eta \models_N \Phi_v^+$ if and only if there is a transition $(\eta \to \eta[v \leftarrow v+1])$ in the state graph $\mathcal{S}$, that is, if and only if the variable v can increase.

3. *We denote by Φ_v^- the following formula:*

$$\Phi_v^- \quad \equiv \quad \bigwedge_{\omega \subset G^{-1}(v)} (\Phi_v^\omega \Longrightarrow K_{v,\omega} < v)$$

Similarly, $\eta \models_N \Phi_v^-$ if and only if the variable v can decrease from the state η in the state graph $\mathcal{S}$.

By the way, in practice, the assertion $assert(\Phi_v^=)$ is often useful from the biological point of view, where $\Phi_v^=$ is obviously defined by: $\Phi_v^= \quad \equiv \quad \bigwedge_{\omega \subset G^{-1}(v)} (\Phi_v^\omega \Longrightarrow K_{v,\omega} = v)$

Our Hoare logic for discrete models of gene networks is then defined by the following rules, where v is a variable of the GRN and $k \in I\!N$.

1. Rules encoding Thomas' discrete dynamics.

Incrementation rule: $\dfrac{}{\{\ \Phi_v^+ \wedge Q[v \leftarrow v+1]\ \}\ v+\ \{Q\}}$

Decrementation rule: $\dfrac{}{\{\ \Phi_v^- \wedge Q[v \leftarrow v-1]\ \}\ v-\ \{Q\}}$

2. Rules coming from Hoare Logic. These rules are very similar to the ones given in Section 2. Obvious rules for the instruction $assert(\Phi)$ and the quantifiers are added:

Assert rule: $$\frac{}{\{\ \Phi \wedge Q\ \}\ assert(\Phi)\ \{\ Q\ \}}$$

Universal quantifier rule: $$\frac{\{P_1\}\ p_1\ \{Q\} \qquad \{P_2\}\ p_2\ \{Q\}}{\{P_1 \wedge P_2\}\ \forall(p_1,p_2)\ \{Q\}}$$

Existential quantifier rule: $$\frac{\{P_1\}\ p_1\ \{Q\} \qquad \{P_2\}\ p_2\ \{Q\}}{\{P_1 \vee P_2\}\ \exists(p_1,p_2)\ \{Q\}}$$

Assignment rule: $$\frac{}{\{Q[v \leftarrow k]\}\ v{:=}k\ \{Q\}}$$

Sequential composition rule: $$\frac{\{P_2\}\ p_2\ \{Q\} \qquad \{P_1\}\ p_1\ \{P_2\}}{\{P_1\}\ p_1;p_2\ \{Q\}}$$

Alternative rule: $$\frac{\{P_1\}\ p_1\ \{Q\} \qquad \{P_2\}\ p_2\ \{Q\}}{\{(e \wedge P_1) \vee (\neg e \wedge P_2)\}\ if\ e\ then\ p_1\ else\ p_2\ \{Q\}}$$

Iteration rule: $$\frac{\{e \wedge I\}\ p\ \{I\}}{\{I\}\ while\ e\ with\ I\ do\ p\ \{\neg e \wedge I\}}$$

Empty program rule: $$\frac{P \Rightarrow Q}{\{P\}\ \varepsilon\ \{Q\}}$$

3. Axioms. These axioms assert that all values stay between their bounds, where v is a variable of the GRN N and $\omega \subset N^{-1}(v)$:

Boundary axioms: $0 \leq v$
$v \leq b_v$
$0 \leq K_{v,\omega}$
$K_{v,\omega} \leq b_v$

Remark 6.2

- *($\Phi_v^+ \Rightarrow v < b_v$) can be derived from the previous rules. Indeed, Φ_v^+ implies that for ω corresponding to the current set of resources, $K_{v,\omega} > v$ and, using the boundary axiom $K_{v,\omega} \leq b_v$, we get $v < b_v$.*
- *Similarly, we have $\Phi_v^- \Rightarrow v > 0$.*

These implications will be used in section **??**.

We will prove in Section 8 that this modified Hoare logic is correct, and that it is complete provided that the path program under consideration contains the weakest loop invariants for all *while* statements. More precisely, the proof strategy called *backward strategy*, already described at the end of Section 2, also applies here: It computes the weakest precondition. Before giving these two proofs, let us show in the next section the usefulness of our genetically modified Hoare logic *via* the formal study of the possible biological functions of a very simple network.

7 Example

In [SOMMA02, MSOI$^+$02] Uri Alon and co-workers have studied the most common in vivo patterns involving three genes. Among them, they have enlightened the "incoherent feedforward loop of type 1". It is composed by a transcription factor a that activates a second transcription

factor c, and both a and c regulate a gene b: a is an activator of b whereas c is an inhibitor of b. There is a "short" positive action of a on b and a "long" negative action *via* c: a activates c which inhibits b. The left hand side of Figure 3 shows such a feedforward loop. Considering that both thresholds of actions of a are equal leads to a boolean network since, in that case, the variable a can take only the value 0 (a has no action) or 1 (a activates both b and c). The right

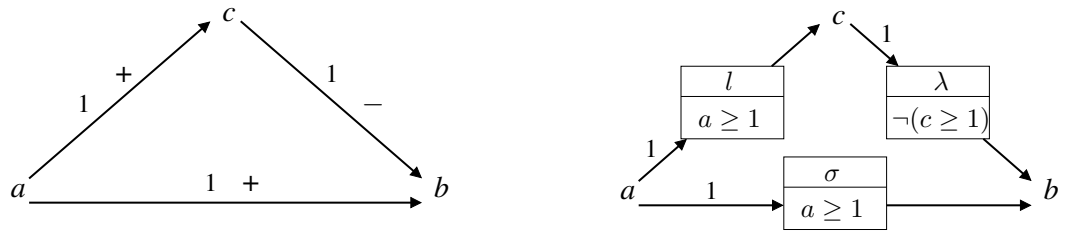

Figure 3: Boolean incoherent feedforward of type 1. At right, graphical representation of the GRN $N = (V, M, E_V, E_M, \mathcal{K})$ with $V = \{a, b, c\}$, the bounds of a, b and c are equal to 1, $M = \{l, \lambda, \sigma\}$, ϕ_l is $(a \geq 1)$, ϕ_λ is $(\neg(c \geq 1))$, ϕ_σ is $(a \geq 1)$. Finally the family of integers is $\{K_{a,\emptyset}, K_{c,\emptyset}, K_{c,\{l\}}, K_{b,\emptyset}, K_{b,\{\sigma\}}, K_{b,\{\lambda\}}, K_{b,\{\sigma,\lambda\}}\}$.

hand side of the figure shows the corresponding GRN with multiplexes: σ encodes the "short" action of a on b, whilst l followed by λ constitute the "long" action.

Several authors, like Uri Alon, consider that if a is equal to 0 for a sufficiently long time, both b and c will also be equal to 0, because b and c need a as a resource in order to reach the state 1. They also consider that the function of this feedforward loop is to ensure a *transitory activity* of b that signals when a has switched from 0 to 1: the idea is that a activates the productions of b and c, and then c stops the production of b.

Here, we take a look at this question *via* four different path programs, and we prove formally that this affirmation is only valid under some constraints on the parameters of the network, and only under the assumption that b starts its activity before c.

Is a transitory production of b possible? As already stated, the function classically associated with the feedforward loop is to ensure a *transitory activity* of b that signals when a has switched from 0 to 1. An interesting question is *under which conditions the previous property is true ?* For example the path program

$$\mathcal{P}_1 \equiv (b+; c+; b-) \tag{1}$$

together with the pre-condition $P \equiv (a = 1 \ \wedge \ b = 0 \ \wedge \ c = 0)$ and the post-condition $Q_0 \equiv \{b = 0\}$, is a possible formalization of the previous property about the behaviour of the feedforward loop. The backward strategy using our genetically modified Hoare logic on this example gives the following successive conditions.

- The weakest precondition obtained through the last instruction "$b-$" is the following conjunction:

$$\Phi_b^- \wedge Q_0[b \leftarrow b-1] \quad \equiv \quad \begin{cases} (\neg\neg(c \geq 1) \wedge \neg(a \geq 1)) \Longrightarrow K_b < b \\ (\neg\neg(c \geq 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma} < b \\ (\neg(c \geq 1) \wedge \neg(a \geq 1)) \Longrightarrow K_{b,\lambda} < b \\ (\neg(c \geq 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma\lambda} < b \\ b - 1 = 0 \end{cases}$$

which simplifies as the conjunction Q_1:

$$Q_1 \equiv \begin{cases} b = 1 \\ ((c \geq 1) \wedge (a < 1)) \Longrightarrow K_b = 0 \\ ((c \geq 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma} = 0 \\ ((c < 1) \wedge (a < 1)) \Longrightarrow K_{b,\lambda} = 0 \\ ((c < 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma\lambda} = 0 \end{cases}$$

- Then, the weakest precondition obtained through the instruction "$c+$" is:

$$\Phi_c^+ \wedge Q_1[c \leftarrow c+1] \equiv \begin{cases} \neg(a \geq 1) \Longrightarrow K_c > c \\ a \geq 1 \Longrightarrow K_{c,l} > c \\ b = 1 \\ ((c+1 \geq 1) \wedge (a < 1)) \Longrightarrow K_b = 0 \\ ((c+1 \geq 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma} = 0 \\ ((c+1 < 1) \wedge (a < 1)) \Longrightarrow K_{b,\lambda} = 0 \\ ((c+1 < 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma\lambda} = 0 \end{cases}$$

which simplifies as Q_2 owing to the boundary axioms and remarks 6.2:

$$Q_2 \equiv \begin{cases} c = 0 \\ a < 1 \Longrightarrow K_c = 1 \\ a \geq 1 \Longrightarrow K_{c,l} = 1 \\ b = 1 \\ a < 1 \Longrightarrow K_b = 0 \\ a \geq 1 \Longrightarrow K_{b,\sigma} = 0 \end{cases}$$

- Lastly, the weakest precondition obtained through the first "$b+$" of the program is:

$$\Phi_b^+ \wedge Q_2[b \leftarrow b+1] \equiv \begin{cases} (\neg\neg(c \geq 1) \wedge \neg(a \geq 1)) \Longrightarrow K_b > b \\ (\neg\neg(c \geq 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma} > b \\ (\neg(c \geq 1) \wedge \neg(a \geq 1)) \Longrightarrow K_{b,\lambda} > b \\ (\neg(c \geq 1) \wedge (a \geq 1)) \Longrightarrow K_{b,\sigma\lambda} > b \\ c = 0 \\ a < 1 \Longrightarrow K_c = 1 \\ a \geq 1 \Longrightarrow K_{c,l} = 1 \\ b + 1 = 1 \\ a < 1 \Longrightarrow K_b = 0 \\ a \geq 1 \Longrightarrow K_{b,\sigma} = 0 \end{cases}$$

which simplifies as Q_3:

$$Q_3 \equiv \begin{cases} a < 1 \Longrightarrow K_{b,\lambda} = 1 \\ a \geq 1 \Longrightarrow K_{b,\sigma\lambda} = 1 \\ c = 0 \\ a < 1 \Longrightarrow K_c = 1 \\ a \geq 1 \Longrightarrow K_{c,l} = 1 \\ b = 0 \\ a < 1 \Longrightarrow K_b = 0 \\ a \geq 1 \Longrightarrow K_{b,\sigma} = 0 \end{cases}$$

Then, using the empty program rule, it comes $P \Longrightarrow Q_3$ *i.e.* $(a = 1 \ \wedge \ b = 0 \ \wedge \ c = 0) \Longrightarrow Q_3$ and after simplification we get the correctness if and only if $K_{b,\sigma\lambda} = 1$ and $K_{c,l} = 1$ and $K_{b,\sigma} = 0$. This proves that, whatever the values of the other parameters, the system can exhibit a transitory production of b in response to a switch of a from 0 to 1.

Is a transitory production of b possible without increasing c? The previous program $\mathcal{P}_1$ is not the only one reflecting a transitory production of b, there may be other realisations of this property. For example one can consider the path program :

$$\mathcal{P}_2 \equiv (b+; b-). \tag{2}$$

With respect to this path program, the weakest precondition obtained through the last instruction "$b-$" is of course Q_1 as previously. Then, the weakest precondition obtained through "$b+$" is:

$$Q_4 \equiv \left\{ \begin{array}{l} b = 0 \\ ((c \geq 1) \wedge (a < 1)) \Longrightarrow ((K_b = 1) \wedge (K_b = 0)) \\ ((c \geq 1) \wedge (a \geq 1)) \Longrightarrow ((K_{b,\sigma} = 1) \wedge (K_{b,\sigma} = 0)) \\ ((c < 1) \wedge (a < 1)) \Longrightarrow ((K_{b,\lambda} = 1) \wedge (K_{b,\lambda} = 0)) \\ ((c < 1) \wedge (a \geq 1)) \Longrightarrow ((K_{b,\sigma\lambda} = 1) \wedge (K_{b,\sigma\lambda} = 0)) \end{array} \right.$$

Q_4 is of course not satisfiable: it implies that each parameter associated with b is both equal to 0 and 1. The path program $(b+; b-)$ is not feasible (inconsistent weakest precondition). Indeed, we retrieve an obvious property of the Thomas' approach: if b has no negative action on itself then the sequence $(b+; b-)$ cannot arise because the resources of b must change in order to switch its direction of evolution.

Another possible path compatible with the path program $(b+, c+, b-)$. Let us notice that, when $K_{b,\sigma\lambda} = 1$, $K_{c,l} = 1$ and $K_{b,\sigma} = 0$, even if the system *can* exhibit a transitory production of b *via* $(b+; c+; b-)$, this does not prevent from some other paths that *do not* exhibit this behaviour. For example the simple path $\mathcal{P}_3 \equiv c+$ leaves b constantly equal to 0, and the Hoare triplet

$$\left\{ \begin{array}{l} a = 1 \ \wedge \ b = 0 \ \wedge \ c = 0 \wedge \\ K_{b,\sigma\lambda} = 1 \ \wedge \ K_{c,l} = 1 \ \wedge \ K_{b,\sigma} = 0 \end{array} \right\} \ c+ \ \left\{ \ b = 0 \ \right\}$$

is satisfied, as the corresponding weakest precondition Q_5 is clearly implied by the precondition.

$$Q_5 \equiv \Phi_c^+ \wedge Q_0[c \leftarrow c+1] \equiv \left\{ \begin{array}{l} c = 0 \\ a = 0 \Longrightarrow K_c = 1 \\ a = 1 \Longrightarrow K_{c,l} = 1 \\ b = 0 \end{array} \right.$$

When a is constantly equal to 1 and when $c = 1$, production of b is impossible. Even worst: when a is constantly equal to 1, once c has reached the level 1, it is impossible for b to increase to 1. We prove this property by showing that the following triplet is inconsistent, whatever the loop invariant I:

$$\left\{ \begin{array}{l} a = 1 \ \wedge \ b = 0 \ \wedge \ c = 1 \wedge \\ K_{b,\sigma\lambda} = 1 \ \wedge \ K_{c,l} = 1 \ \wedge \ K_{b,\sigma} = 0 \end{array} \right\} \underbrace{\textit{while } b<1 \textit{ with } I \textit{ do } \exists(b+, b-, c+, c-)}_{\mathcal{P}_4} \left\{ \ b=1 \ \right\} \tag{3}$$

The subprogram $\exists(b+, b-, c+, c-)$ reflects the fact that a stays constant but b or c evolves. The *while* statement allows b and c to evolve freely until b becomes equal to 1.

Applying the Iteration rule, I has to satisfy:

- $\neg(b < 1) \wedge I \Longrightarrow (b = 1)$
  This property is trivially satisfied whatever the assertion I, due to the boundary axioms.

- $\{b < 1 \wedge I\}\ \exists(b+, b-, c+, c-)\ \{I\}$
  We apply the existential quantifier rule, which gives the following weakest precondition:

  $$Q_6 \ \equiv\ (\Phi_b^+ \wedge I[b \leftarrow b+1]) \vee (\Phi_b^- \wedge I[b \leftarrow b-1]) \vee (\Phi_c^+ \wedge I[c \leftarrow c+1]) \vee (\Phi_c^- \wedge I[c \leftarrow c-1])$$

  Consequently I can be any assertion such that

  $$(b = 0 \wedge I) \Longrightarrow Q_6$$

Let us denote P the precondition of the path program $\mathcal{P}_4$. Applying the empty program rule, it comes that I must also satisfy $P \Longrightarrow I$. So, because $P \Longrightarrow (b = 0)$, we have $P \Longrightarrow (b = 0 \wedge I)$, which, in turn implies Q_6. Moreover, let us remark that

$$Q_6 \ \Longrightarrow\ (\Phi_b^+ \vee \Phi_b^- \vee \Phi_c^+ \vee \Phi_c^-)$$

Consequently, if the Hoare triple 3 is correct, then:

$$P \ \Longrightarrow\ (\Phi_b^+ \vee \Phi_b^- \vee \Phi_c^+ \vee \Phi_c^-)$$

which is impossible because, if P is satisfied, then:

- Φ_b^+ is false, as $a = 1$, $c = 1$ and $K_{b,\sigma} = 0$ (indeed,Φ_b^+ implies $a = 1 \wedge c = 1 \Rightarrow K_{b,\sigma} > 0$)
- Φ_b^- is false, as $b = 0$ (Φ_b^- implies $b > 0$)
- Φ_c^+ is false, as $c = 1$ (Φ_c^+ implies $c < 1$)
- Φ_c^- is false, as $a = 1$, $c = 1$ and $K_{c,l} = 1$ (Φ_c^- implies $a = 1 \wedge c = 1 \Rightarrow K_{c,l} < 1$).

So, we have formally proved that when a is constantly equal to 1, once c has reached the level 1, it is impossible for b to increase to 1.

8 Partial Correctness and Completeness

"Partial" has to be understood here as "Assuming that the *while* loops terminate", as usual in Hoare logic.

8.1 Correctness

Correctness of our modified Hoare logic means that: if $\vdash \{P\}\ p\ \{Q\}$ according to the inference rules of Section 6, then the Hoare triple $\{P\}\ p\ \{Q\}$ is semantically satisfied according to Definition 5.3, i.e.: for all $\eta \in S$ (S being the set of states of $\mathcal{S}$) such that $\eta \models_N P$ there exists $E \subset S$ such that $\eta \overset{p}{\rightsquigarrow} E$ and $\forall \eta' \in E,\ \eta' \models_N Q$.

The proof is made as usual by induction on the proof tree of $\vdash \{P\}\ p\ \{Q\}$. Hence, we have to prove that each rule of Section 6 is correct. Here we develop only the *Incrementation rule* and the *Sequential composition rule* since the correctness of the other inference rules is either similar (*Decrementation rule*) or trivial (*Assert rule*, *quantifier rules*, *Assignment rule* and *Empty program rule*) or standard in Hoare Logic (*Alternative rule* and *Iteration rule*). Let us note that the correctness of the *Sequential composition rule* is neither trivial nor standard because its semantics is enriched to cope with the quantifiers.

Let N be a GRN and let η be any state of the associated state space S:

Incrementation rule: $\dfrac{}{\{\ \Phi_v^+ \ \wedge\ Q[v \leftarrow v+1]\ \}\ v+\ \{Q\}}$ (where v is a variable of the GRN)

From Definition 5.3, the hypothesis is:

$\boxed{H}$ $\eta \models_N \Phi_v^+$ and $\eta \models_N Q[v \leftarrow v+1]$

and we have to prove the conclusion:

$\boxed{C}$ there exists $E \subset S$ such that $\eta \overset{v+}{\rightsquigarrow} E$ and $\forall \eta' \in E,\ \eta' \models_N Q$

Let us choose $E = \{\eta'\}$ with $\eta' = \eta[v \leftarrow \eta(v)+1]$. From Notation 6.1, the hypotesis $\eta \models_N \phi_v^+$ is equivalent to $(\eta \rightarrow \eta') \in \mathcal{S}$, which in turn, according to Definition 5.2, implies $\eta \overset{v+}{\rightsquigarrow} \{\eta'\}$. Hence, it only remains to prove that $\eta' \models_N Q$, which results from the hypothesis $\eta \models_N Q[v \leftarrow v+1]$. □

Sequential composition rule: $\dfrac{\{P_2\}\ p_2\ \{Q\} \qquad \{P_1\}\ p_1\ \{P_2\}}{\{P_1\}\ p_1;p_2\ \{Q\}}$

From Definition 5.3, we consider the following three hypotheses:

$\boxed{H_1}$ for all $\eta_1 \in S$ such that $\eta_1 \models_N P_1$ there exists E_1 such that $\eta_1 \overset{p_1}{\rightsquigarrow} E_1$ and $\forall \eta' \in E_1,\ \eta' \models_N P_2$

$\boxed{H_2}$ for all $\eta_2 \in S$ such that $\eta_2 \models_N P_2$ there exists E_2 such that $\eta_2 \overset{p_2}{\rightsquigarrow} E_2$ and $\forall \eta'' \in E_2,\ \eta'' \models_N Q$

$\boxed{H_3}$ $\eta \models_N P_1$

and we have to prove the conclusion:

$\boxed{C}$ there exists $E \subset S$ such that $\eta \overset{p_1;p_2}{\rightsquigarrow} E$ and $\forall \eta'' \in E,\ \eta'' \models_N Q$

Let us arbitrarily choose a set E_1 such that $\eta \overset{p_1}{\rightsquigarrow} E_1$ and $\forall \eta' \in E_1,\ \eta' \models_N P_2$ (we know that E_1 exists from $\boxed{H_1}$ and $\boxed{H_3}$).

For each $\eta' \in E_1$, we similarly choose a set $E_2^{\eta'}$ such that:

$\eta' \overset{p_2}{\rightsquigarrow} E_2^{\eta'}$ and $\forall \eta'' \in E_2^{\eta'},\ \eta'' \models_N Q$ (we know that the family $\{E_2^{\eta'}\}_{\eta' \in E_1}$ exists from $\boxed{H_2}$ and the fact that $\eta' \models_N P_2$ for all $\eta' \in E_2$)

Let $E = (\bigcup_{\eta' \in E_1} E_2^{\eta'})$, we have: $\eta \overset{p_1;p_2}{\rightsquigarrow} E$ from Definition 5.2 and $\forall \eta'' \in E,\ \eta'' \models_N Q$ (from the way the union is built). □

8.2 Weakest precondition

Completeness of Hoare logic would be of course defined as follows: If the Hoare triple $\{P\}\ p\ \{Q\}$ is satisfied (according to Definition 5.3) then $\vdash \{P\}\ p\ \{Q\}$ (using the inference rules of Section 6 as well as first order logic and proofs on integers).

Obviously, as such, Hoare logics cannot be complete because, as already mentioned for classical Hoare logic, finding the weakest loop invariants is undecidable and there is no complete logic on integers (Gödel). So, following Dijkstra [Dij75], we prove completeness under the assumptions that the loop invariants of all *while* statements are weakest invariants and that the needed properties on integers are admitted. We adopt the strategy that computes the weakest precondition and we will prove the following theorem:

Theorem 8.1 (Dijkstra theorem on the genetically modified Hoare logic) *A* GRN N *and a Hoare triple* $\{P\}\ p\ \{Q\}$ *being given, the backward strategy defined at the end of Section 2, with the inference rules of Section 6, computes the* weakest precondition P_0 *just before the last inference that uses the* Empty program rule.
It means that: if $\{P\}\ p\ \{Q\}$ *is satisfied, then* $P \Rightarrow P_0$ *is satisfied.*

This theorem has an obvious corrolary:

Corollary 8.2 *A* GRN N *being given, our modified Hoare logic is complete under the assumption that all given loop invariants are the weakest ones and that the needed properties on integers are established.*

Proof of the corollary: if $\{P\}\ p\ \{Q\}$ is satisfied, then, from the Dijkstra theorem above, there is a proof tree that infers the Hoare triple if there is a proof tree for the property $P \Rightarrow P_0$ (which is semantically satisfied because P_0 is the weakest precondition). First order logic being complete and properties on integers being axiomatically assumed, the proof tree for $P \Rightarrow P_0$ exists. □

Proof of Dijkstra theorem:
Under the hypotheses that all loop invariants are minimal and that the Hoare triple $\{P\}\ p\ \{Q\}$ is satisfied, i.e., under the hypotheses:

$\boxed{H_1}$ for all η satisfying P, there exists E such that $\eta \overset{p}{\rightsquigarrow} E$ and for all $\eta' \in E$, η' satisfies Q

$\boxed{H_2}$ for all *while* statements of p, the corresponding loop invariant I is the weakest one

one has to prove the conclusion:

$\boxed{C}$ $P \Rightarrow P_0$ is satisfied, where P_0 is the precondition computed by the proof of $\{P\}\ p\ \{Q\}$ according to the backward strategy with the inference rules of Section 6.

The proof is done by structural induction according to the backward strategy on p.

- If p is of the form *while* e *with* I *do* p', then, by construction of the backward strategy, applying the *Iteration rule*, we get $P_0 = I$, and the conclusion results immediately from $\boxed{H_2}$.

- If p is of the form $v+$, then the only set E such that $\eta \overset{v+}{\rightsquigarrow} E$ is $E = \{\eta[v \leftarrow v+1]\}$. The hypothesis $\boxed{H_1}$ becomes:

  $\boxed{H_1}$ for all η satisfying P, $\eta' = \eta[v \leftarrow v+1]$ satisfies Q and $\eta \rightarrow \eta'$ is a transition of $\mathcal{S}$

  and from the *Incrementation rule*, the conclusion becomes:

  $\boxed{C}$ $P \Rightarrow (\Phi_v^+ \wedge Q[v \leftarrow v+1])$ is satisfied.

  So, $\boxed{H_1} \Rightarrow \boxed{C}$ straightforwardly results from the definition of Φ_{v^+} (Notation 6.1) and we do not use $\boxed{H_2}$.

- If p is of the form $p_1; p_2$, then we firstly inherit the two structural induction hypotheses:

  $\boxed{H_3}$ for all assertions P' and Q', if $\{P'\}\ p_1\ \{Q'\}$ is satisfied then $P' \Rightarrow P_1$ is satisfied, where P_1 is the precondition computed from Q' *via* the backward strategy

  $\boxed{H_4}$ for all assertions P'' and Q'', if $\{P''\}\ p_2\ \{Q''\}$ is satisfied then $P'' \Rightarrow P_2$ is satisfied, where P_2 is the precondition computed from Q'' *via* the backward strategy

  Moreover the hypothesis $\boxed{H_1}$ becomes (Definition 5.2):

$\boxed{H_1}$ for all η satisfying P, there exists a family of state sets $\mathcal{F} = \{E_e\}_{e \in F}$ such that $\eta \overset{p_1}{\rightsquigarrow} F$ and $e \overset{p_2}{\rightsquigarrow} E_e$ for all $e \in F$ and for all $\eta' \in E = (\bigcup_{e \in F} E_e)$, η' satisfies Q

Lastly, from the *Sequential composition rule*, the conclusion becomes:

$\boxed{C}$ $P \Rightarrow P_1$ is satisfied, where P_1 is the weakest precondition of $\{\cdots\}\ p_1\ \{P_2\}$, P_2 being the weakest precondition of $\{\cdots\}\ p_2\ \{Q\}$.

From $\boxed{H_4}$ (with $Q'' = Q$) it results that all the states $e \in F$ of hypothesis $\boxed{H_1}$ satisfy P_2. Consequently $\{P\}\ p_1\ \{P_2\}$ is satisfied. Thus, from $\boxed{H_3}$ (with $Q' = P_2$ and $P' = P$) it comes $P \Rightarrow P_1$, which proves the conclusion.

- Similarly to the correctness proof, we do not develop here the other cases of the structural induction. They are either similar to already developed cases (*Decrementation rule*) or trivial (*Assert rule*, *quantifier rules*, and *Assignment rule*) or standard in Hoare Logic (*Alternative rule*).

This ends the proof. □

9 Discussion

The cornerstone of the modeling process lies, whatever the application domain, in the determination of parameters. In this paper, we proposed an approach for exhibiting constraints on parameters of gene network models, that relies on the adaptation of the Hoare logic, initially designed for proofs of imperative programs. It leads to several questions about its usability and implementations.

9.1 Language issues

The path language is a way to describe formally the specification of correct models of gene networks. Classically, the specifications can be expressed in temporal logics, like CTL and LTL, which also allows the modeler to take into account behavioral information. But even if there exists some links between path language and temporal logics, these formal languages (temporal logics and path language) are not comparable: some properties can be expressed in the path language and not in classical temporal logics and converselly.

- On the one hand, in the path program, the assignment instruction allows one to express a knock-out of a gene ($v := 0$). Such knock-out of a gene is not expressible in CTL or LTL.
- On the other hand, CTL or LTL is able to express properties on infinite cyclic traces. Such properties on infinite traces would be expressed in the path language by a program which does not terminate, and consequently, the post-condition would not make sense.

Nevertheless, a succession of incrementation/decrementation instructions corresponds to a property that can be expressed in the CTL language. For example, if one knows the starting point, say $(v_1 = 1 \wedge v_2 = 0)$, the path program $v_1+; v_2+; v_1-$ corresponds to the formula $EX(v_1 = 2 \wedge EX(v_2 = 1 \wedge EX(v_1 = 1 \wedge v_2 = 0)))$. Correctness of this program path with the precondition $(v_1 = 1 \wedge v_2 = 0)$ becomes equivalent to verify that the CTL formula $(v_1 = 1 \wedge v_2 = 0) \Rightarrow EX(v_1 = 2 \wedge EX(v_2 = 1 \wedge EX(v_1 = 1 \wedge v_2 = 0)))$ is true in all possible states. More generally, the path language is well suited for sequential properties whereas CTL can express non sequential ones.

```
# initializing
T3:=1; T4:=1; d3:=1; d2:=0; gi:=0; gp:=0; gt:=0; tr:=0;
# evolutions
gi+; d2+; T3+; tr+; T3+; gp+; d3-; gt+;
```

$$\begin{pmatrix} (T3=1) & \wedge \\ (T4=1) & \wedge \\ (d3=1) & \wedge \\ (d2=0) & \wedge \\ (gi=0) & \wedge \\ (gp=0) & \wedge \\ (gt=0) & \wedge \\ (tr=0) & \end{pmatrix} \Rightarrow \begin{pmatrix} EX((gi=1)\wedge \\ \quad EX((d2=1)\wedge \\ \quad\quad EX((T3=2)\wedge \\ \quad\quad\quad EX((tr=1)\wedge \\ \quad\quad\quad\quad EX((T3=3)\wedge \\ \quad\quad\quad\quad\quad EX((gp=1)\wedge \\ \quad\quad\quad\quad\quad\quad EX((d3=0)\wedge \\ \quad\quad\quad\quad\quad\quad\quad EX(gt=1)))))))) \end{pmatrix}$$

Figure 4: A path program (top) with its corresponding CTL formula (bottom)

In the path language, invariants of *while* loops are mandatory: Hoare logic is able to prove a program with *while* statements only if invariants are given. In other words, the entire information that the Hoare logic needs to perform the proof, is given by invariants. Unfortunately, invariants are difficult to devise. Thus the *while* statements are often used in proofs by refutation, where the proof is done for each possible invariant, see our example of section **??**.

9.2 Plateform issues

The Hoare logic for gene networks has been designed in order to support a software which aims at helping the determination of parameters of models of gene networks. We have already done its proof of feasability. Indeed, after having developped a prototype named `SMBioNet` which enumerates all possible valuations of parameters and retains only those which are coherent with a specified temporal logic formula, we developped a new prototype called `WP-SMBioNet` [Kha10], which uses a path program and the Weakest Precondition calculus (backward strategy) to produce constraints on parameters.

In order to compare both approaches (CTL formulae *versus* path programs), we consider a property which can be expressed in both temporal logic CTL and path program. When modeling the biological system triggering the tail resorption during the metamorphose of tadpole, see [TTB+07] and references therein, the expression profiles of [LB77] can be translated into a path program, see Fig 4, which in turn can be translated into an equivalent CTL formula. For this example, whereas `SMBioNet` needs more than 3 hours to selects among all possible parameterizations those which lead to a dynamics coherent with the CTL formula, `WP-SMBioNet` needs only 10 seconds (on the same computer) to construct the constraints on the parameters. If we ask the enumeration of the parameters satisfying the constraints (using Choco [cT10]), the total search time is about 2 minutes. This example shows that the Hoare logic can speed up the determination of coherent parameterizations.

We can notice that the complexity of the weakest precondition calculus is linear with the number of instructions in the path program, and does not depend on the size of gene regulatory networks: each node of the syntaxic tree of the program is visited only once. At the opposite, the CTL model checking algorithm depends on the size of the network. Thus, the use of path program instead of CTL formula leads to postpone the enumeration step which then can use the constraints on parameters to cut down drastically the set of parameterizations to consider.

A software plateform dedicated to analysis of gene regulatory networks, should have to

combine different technics. Indeed constraints solving technics are necessary to enumerate parameters or give counter-examples, theorem prover can be also useful to get strategies for proofs by refutation, and model checking technics and Hoare logic precondition calculus should be combined in order to give very efficient algorithm. As already noted, it seems natural to use Hoare logic when the behavioural specification focuses on a finite time horizon, whereas model checking is natural when the temporal specification expresses global properties on infinite traces.

It would be interesting to complete this plateform with some improved features. From a theoretical point of view, one could also develop approaches to help finding loop invariants. To build them, it seems possible to adapt the iterative approach adopted in ASTREE [CCF$^+$05] but in another context (abstract interpretation [CC04]): Pragmatically one begins with a simple invariant I, then one tries to make the proof and completes iterativelly and partially the invariant. From an application point of view, specifications often stem from DNA profiles, it would be valuable to develop a program that automatically produces path programs from DNA chips data. Two questions emerge: the choice of thresholds on which is based the discretization of expression levels, and the determination of good time steps. These questions are out of the scope of this article. They mainly rely on biological expertise and experimental conditions.

Acknowledgment

We are grateful to Alexander Bockmayr and Heike Siebert for fruitful discussions and comments on this paper. The authors thank the French National Agency for Research (ANR-10-BLANC-0218 BioTempo project) for its support. This work has also been supported by the CNRS PEPII project "*CirClock*".

References

[BCRG04] G. Bernot, J.-P. Comet, A. Richard, and J. Guespin. Application of formal methods to biological regulatory networks: Extending Thomas' asynchronous logical approach with temporal logic. *Journal of Theoretical Biology*, 229(3):339–347, 2004.

[BG01] A. Blass and Y. Gurevich. Inadequacy of computable loop invariants. *ACM Transactions on Computational Logic*, 2(1), 2001.

[BPGT01] C. Boileau, J. Prados, J. Geiselmann, and L. Trilling. Using constraint programming for learning from experiments transcriptional activation and the geometry of DNA. In T. Schiex L. Duret, C. Gaspin, editor, *Actes des Journes Ouvertes Biologie Informatique Mathmatiques (JOBIM), Toulouse*, 2001.

[CC04] P. Cousot and R. Cousot. *Building the Information Society*, chapter Basic Concepts of Abstract Interpretation., pages 359–366. Kluwer Academic Publishers, 2004.

[CCF$^+$05] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Min, D. Monniaux, and X Rival. The ASTRE analyser. In M. Sagiv, editor, *ESOP 2005 The European Symposium on Programming*, number 3444 in LNCS, pages 21–30. Springer, 2005.

[Cor08] Fabien Corblin. *Conception et mise en oeuvre d'un outil déclaratif pour l'analyse des réseaux génétiques discrets*. PhD thesis, Université Joseph-Fourier - Grenoble I, 12 2008.

[cT10] choco Team. choco: an open source java constraint programming library. Research report 10-02-INFO, Ecole des Mines de Nantes, 2010.

[CTF+09] F. Corblin, S. Tripodi, E. Fanchon, D. Ropers, and L. Trilling. A declarative constraint-based method for analyzing discrete genetic regulatory networks. *Biosystems*, 98(2):91–104, 2009.

[Dij75] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18:453–457, August 1975.

[FCT+04] E. Fanchon, F. Corblin, L. Trilling, B. Hermant, and D. Gulino. Modeling the molecular network controlling adhesion between human endothelial cells: Inference and simulation using constraint logic programming. In *CMSB*, pages 104–118, 2004.

[Hat74] W.S. Hatcher. A semantic basis for program verification. *J. of Cybernetics*, 4(1):61–69, 1974.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–585, oct 1969.

[KCRB09] Z. Khalis, J.-P. Comet, A. Richard, and G. Bernot. The SMBioNet method for discovering models of gene regulatory networks. *Genes, Genomes and Genomics*, 3(special issue 1):15–22, 2009.

[Kha10] Z. Khalis. *Logique de Hoare et identification formelle des paramtres d'un rseau gntique*. PhD thesis, University of Evry-Val d'Essonne, 2010.

[LB77] J. Leloup and M. Buscaglia. La triiodothyronine, hormone de la mtamorphose des amphibiens. *CR Acad. Sci.*, 284:2261–2263, 1977.

[MGCLG07] D. Mateus, J.-P. Gallois, J.-P. Comet, and P. Le Gall. Symbolic modeling of genetic regulatory networks. *Journal of Bioinformatics and Computational Biology*, 5(2B):627–640, 2007.

[MSOI+02] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298:824–827, 2002.

[SOMMA02] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31:64–68, 2002.

[Td90] R. Thomas and R. d'Ari. *Biological Feedback*. CRC Press, 1990.

[Tho91] R. Thomas. Regulatory networks seen as asynchronous automata : A logical description. *J. theor. Biol.*, 153:1–23, 1991.

[TK01] R. Thomas and M. Kaufman. Multistationarity, the basis of cell differentiation and memory. II. logical analysis of regulatory networks in terms of feedback circuits. *Chaos*, 11:180–195, 2001.

[TTB+07] S. Troncale, R. Thuret, C. Ben, N. Pollet, J.-P. Comet, and G. Bernot. Modelling of the TH-dependent regulation of tadpole tail resorption. *Journal of Biological Physics and Chemistry*, 7(2):45–50, 2007.